Parallel R

ANF R

Vincent Miele

CNRS

07/10/2015

Plan

└Context

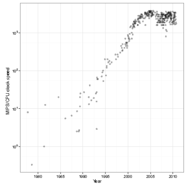
Large computing problems in the era of high-throughput data

- huge data size
- wide solution spaces (combinatorics. . .)
- expensive algorithms (MCMC. . .)
- (memory requirements cannot be met by the memory of a single system)

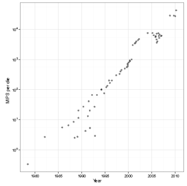
└Context

Static clock speed for a single CPU* around 3.4 Ghz (beyond, the CPU melt)

*(processing unit)



Due to the rise of multi-core machines, the computational power per die has still been increasing (Moore's law)
Moreover, clusters are widely available



- before 2003 : expecting more power
- after 2003 : thinking parallel

Principles

Parallel algorithms **design** and **implementation** is the way to take advantage of multiprocessors architecture. (see ▸ Introduction to parallel computing, Gramma et al, Addison Wesley)

1. decomposition into *tasks* (indivisible units executed in parallel) of various size
2. listing the dependencies between tasks and evaluate their size
3. mapping tasks to *process* : scheduling/synchronizing to respect dependencies, avoid waiting time and minimize total time
4. distributing the input/output
5. managing access on common data

└Traditional paradigms and languages

On a *shared memory* computer (like multi-cores), communication is implicit since all the memory is accessible to all the processes.

ooo From low (`threads`) to medium (`openMP, Intel TBB`) to high level interfaces (`Python multiprocessing`).

On a *distributed memory* computer (like clusters), the programmer is responsible for refactoring his code and add explicit operations for managing concurency, assuming a partitionned adress space.

ooo Low-level `MPI` standard.

Plan

embarrassingly parallel computations in R

Running large chunks of **independent** computations in parallel
≡ *coarse grain* parallelism
≡ embarrassingly parallel

Basic model is :

1. a *master* R process
2. starting up *m worker* processes (not threads)
3. spliting and sending the tasks along with the associated data
4. waiting for all the workers to complete their tasks
5. the *master* R collects the results
6. shut down the worker processes

└*embarrassingly parallel* computations in R

Running large chunks of **independent** computations in parallel
≡ *coarse grain* parallelism
≡ ▸ embarrassingly parallel

Basic model is :

1. a *master* R process
2. starting up *m worker* processes (not threads)
3. spliting and sending the tasks along with the associated data
4. waiting for all the workers to complete their tasks
5. the *master* R collects the results
6. shut down the worker processes

What about the communication between *master* and *workers* ?

└ the snow heritage

▸ SNOW

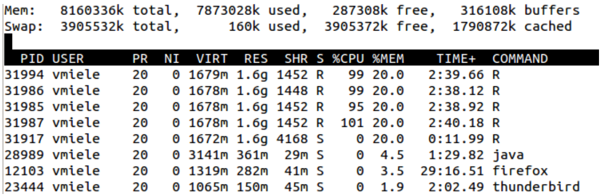
ooo "Simple Networks Of Workstations"
ooo 10 years of activity
ooo on Linux, Mac OS X and Windows
ooo via system("Rscript") or similar to launch a new process with an identical R installation
ooo uses different transport mechanisms to communicate, in particular *sockets* (on a single machine) and MPI (on a cluster, via package ▸ Rmpi)

But, warning, network traffic can lead to overheads...

the multicore heritage

multicore

ooo more recent (2009) but based on well established concepts
ooo only on Linux and Mac OS X
ooo via fork system call that creates complete copy of the master process (only the PID is different)
ooo copied *workers* will share memory pages with the master until modified so forking is very fast (copy-on-write mechanism)



```
Mem:   8160336k total,  7873028k used,   287308k free,   316108k buffers
Swap:  3905532k total,      160k used,  3905372k free,  1790872k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
31994 vmiele    20   0 1679m 1.6g 1452 R   99 20.0   2:39.66 R
31986 vmiele    20   0 1678m 1.6g 1448 R   99 20.0   2:38.12 R
31985 vmiele    20   0 1678m 1.6g 1452 R   95 20.0   2:38.92 R
31987 vmiele    20   0 1678m 1.6g 1452 R  101 20.0   2:40.18 R
31917 vmiele    20   0 1672m 1.6g 4168 S    0 20.0   0:11.99 R
28989 vmiele    20   0 3141m 361m  29m S    0  4.5   1:29.82 java
12103 vmiele    20   0 1319m 282m  41m S    0  3.5  29:16.51 firefox
23444 vmiele    20   0 1065m 150m  45m S    0  1.9   2:02.49 thunderbird
```

Implicit communication through the shared memory

└the Rmpi heritage (in brief)

▸ Rmpi

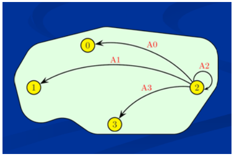
ooo R bindings above MPI ▸ Message Passing Interface

ooo the same code is executed by multiple processes, with flags to assign tasks to workers

```
If I am process 0 {do something; send data to others} else {wait data from 0; do somethingelse}
```

ooo explicit communications ≡ *fine grain* between processes, i.e. the programmer need to implement the data tranfers

- which process is sending ? which process is going to receive ?
- where is the data to be sent ? what kind of data ? how much ?
- where should the data be left on the receiving process ?



A low-level paradigm with a high-level langage. . . tortuous !

Plan

└The `parallel` package

Package parallel was first included in R 2.14.0. It builds on the work done for CRAN packages multicore (Urbanek, 2009-present) and snow (Tierney et al., 2003-present) and provides drop- in replacements for most of the functionality of those packages, with integrated handling of random-number generation. ▸ parallel

Data Analysis in the Distributed World
Parallel R
O'REILLY®
Q. Ethan McCallum & Stephen Weston

snow-like

- makeCluster(. . . , type=“PSOCK”) + stopCluster
- parApply
- clusterEvalq + clusterExport

TP Exercise(s) 1+2

ooo both stdout() and stderr() of the workers are redirected, by default being discarded but they can be logged using the outfile option

multicore-like

▸ mclapply

ooo both stdout() and stderr() in parallel (could lead to conflicts and stranges words/sentences)

ooo You can use mclapply via parLapply using the fork backend. parLapply is the most general interface (but warning of data copies !)

▸ makeCluster(. . . , type="FORK") + stopCluster

▸ parLapply, parApply. . .

TP Exercise(s) 1+2

▸ mcparallel (a submit operation) and ▸ mccollect (a wait/check operation)

TP Exercise(s) 3

mpi-like

▸ makeCluster(. . . , type="MPI") + stopCluster

ooo to take advantage of clusters and to get large distributed RAM
ooo package `Rmpi` must be installed (and conseq. `MPI` on your system)
ooo associated with a batch queueing system
Complicated... probably more appropriate to go to a low level language (and use `MPI`)

└ The foreach+doParallel packages

▸ doParallel

ooo acts as an interface between foreach and the parallel package of R
ooo integrates the foreach flexibility and the ▸ combine possibility
ooo enables ▸ nested foreach loops

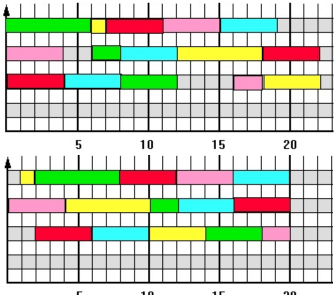
TP Exercise(s) 4

`foreach` versus `*apply` : a troll ?

Plan

└Load balancing

How to schedule the tasks (i.e. assign the data to be treated) ? A very difficult problem !
ooo static and blind ≡ roughly dividing by the number of processes
ooo static but optimized ≡ when knowing the task expected duration, using ad-hoc optimization algorithms (Longest processing time first (LPT) rule (Graham, 1966))



ooo dynamic ≡ sending chunks of tasks to unloaded processes, one by one

Parallel R - the hard way

Load balancing

▸ mclapply(. . . , mc.preschedule =FALSE)

▸ parLapplyLB

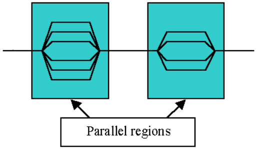
ooo warning, a process is created each time (communication) : possible overhead

Better version : ▸ parLapplyLB

TP Exercise(s) 5+6

Amdahl's law : towards the best speedup

The theoretical maximum speedup is given by $\frac{1}{\alpha+(1-\alpha)/c}$, where α is the fraction of time spent in the sequential part and c the nomber of processes



If $\alpha = 10\%$ then the maximum speedup is 10 even if $c \to \infty$...

└Amdahl's law : towards the best speedup

What could prevent you from reaching the Amdahl's law ?
ooo all operations that leads to starting R processes may be expessive (see section "Parallel R - for real")
ooo fo slow-like functions, the cost of data tranfer (especially on clusters) may be prohibitive, so tranfer no more than the required data :

The key point is ▸ the environments in R !
All that belongs to `.GlobalEnv` needs to be explicitely transfered with ▸ clusterExport.

TP Exercise(s) 7

└Amdahl's law : towards the best speedup

What could prevent you from reaching the Amdahl's law ?
ooo all operations that leads to starting R processes may be expessive (see section "Parallel R - for real")
ooo fo slow-like functions, the cost of data tranfer (especially on clusters) may be prohibitive, so tranfer no more than the required data :

The key point is ▸ the environments in R !
All that belongs to `.GlobalEnv` needs to be explicitely transfered with ▸ clusterExport.

TP Exercise(s) 7

All that belongs to any other environments is serialized (copied as a whole). When a function is defined in a block (typically a method of a S4 class), it has its own environment. . . so all the data defined inside a function are serialized (even if not necessary). . . so move the function to `.GlobalEnv` ! And consider that the parameters of a function are serialized.

TP Exercise(s) 8

└Pseudo-random number generation

Worker processes might get the same seed because a workspace containing `.Random.seed` was restored or the random number generator has been used before forking. . .

Package parallel contains an implementation of the ideas of L'Ecuyer et al. (2002) that is more efficient than R's default "Mersenne-Twister" RNG : `RNGkind("L'Ecuyer-CMRG")`

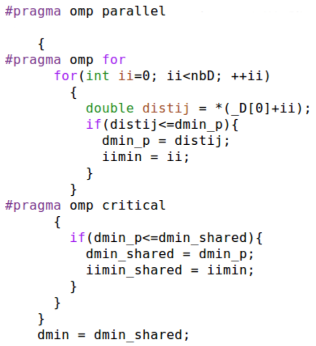
▸ more details

Parallel R - the hard way

openMP

It is recommended to recode hotspot R functions (see section “Optimizing R code”) in `C/C++/Fortran`. Moreover, it is possible to introduce shared memory parallelism with `openMP` standard

ooo widely supported and documented (see ▸ cours openMP)
ooo adding `openMP` directives to help to compiler to parallelize (in particular loops)
ooo requires a specific ▸ Makevars in `pkg/src` to add the `-fopenmp` compiler flag



```
#pragma omp parallel

    {
#pragma omp for
      for(int ii=0; ii<nbD; ++ii)
        {
          double distij = *(_D[0]+ii);
          if(distij<=dmin_p){
            dmin_p = distij;
            iimin = ii;
          }
        }
#pragma omp critical
      {
        if(dmin_p<=dmin_shared){
          dmin_shared = dmin_p;
          iimin_shared = iimin;
        }
      }
    }
    dmin = dmin_shared;
```

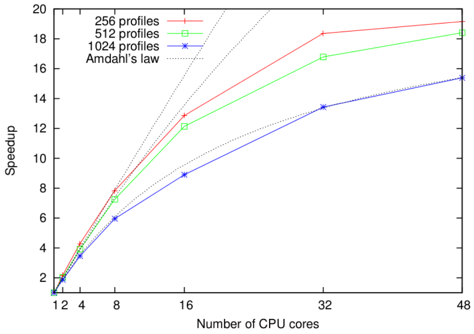
Plan

Parallel R - for real

▸ cghseg

Guillem Rigaill, Vincent Miele and Franck Picard. *Fast and parallel Algorithm for Population-Based Segmentation of Copy-Number Profiles.* LNCS 2014.

DEMO 1 (cghseg in action)



Plan

Others things...

pbdR ?
Rcppparallel ?
http ://www.hpl.hp.com/research/systems-research/R-workshop